Embedded Policing and Policy Enforcement Approach for Future Secure IoT Technologies

Embedded Policing and Policy Enforcement Approach for Future Secure IoT Technologies

Fahad Siddiqui*, Matthew Hagan*, Sakir Sezer*

** Center for Secure Information Systems (CSIT), Queen's University Belfast, UK*

Keywords: FPGA, MPSoC, Root-of-trust, ARM TrustZone.

Abstract

The Internet of Things (IoT) holds great potential for productivity, quality control, supply chain efficiencies and overall business operations. However, with this broader connectivity, new vulnerabilities and attack vectors are being introduced, increasing opportunities for systems to be compromised by hackers and targeted attacks. These vulnerabilities pose severe threats to a myriad of IoT applications within areas such as manufacturing, healthcare, power and energy grids, transportation and commercial building management. While embedded OEMs offer technologies, such as hardware *Trusted Platform Module* (TPM), that deploy strong *chain-of-trust* and *authentication* mechanisms, still they struggle to protect against vulnerabilities introduced by vendors and end users, as well as additional threats posed by potential technical vulnerabilities and zero-day attacks. This paper proposes a pro-active policy-based approach, enforcing the principle of least privilege, through hardware *Security Policy Engine* (SPE) that actively monitors communication of applications and system resources on the system communication bus (ARM AMBA-AXI4). Upon detecting a policy violation, for example, a malicious application accessing protected storage, it counteracts with predefined mitigations to limit the attack. The proposed SPE approach widely complements existing embedded hardware and software security technologies, targeting the mitigation of risks imposed by unknown vulnerabilities of embedded applications and protocols.

1 Introduction

The IoT market has grown significantly, with a massive range of IoT products reaching further into different areas, including home user's private lives, the enterprise, as well as deployment to control critical manufacturing and industrial appliances [1]. Indeed, estimates from Gartner predict that IoT will proliferate to 20.5 Billion Devices by 2020 [2]. Simultaneously, so too has the opportunity for malicious actors to take advantage of IoT devices, exposing them to a myriad of security risks [3]. These security risks may include compromising the user's privacy, damaging industrial applications, steal information from an enterprise, or using the IoT's own computing resources to launch attacks elsewhere, as seen in the Mirai botnet [4].

The required level of security for IoT cannot be defined generically due to highly versatile application markets such as mobile, consumer electronics, medical and industrial control solutions. IoT security is a challenging research problem, with trust issues at every level, from authentication of the user, communication between interconnected devices, handling of sensitive data and the potential for malware and physical attacks to compromise any aspect of the ecosystem [5].
With such a wide range of potential attacks, efforts must be made at all levels of design to incorporate security, to ensure trustworthiness of devices, and to prevent compromise of user data and device. It includes design philosophy, software and hardware architectures, choice of components used and even where manufacturing takes place [6]. In addition, relevant secure practices must be extended to and enforced at every layer within the IoT ecosystem, which typically comprises of the following components:

- **Cloud:** Providing data storage and overarching computing services.
- **Gateway:** Interfacing the edge to the cloud component.
- **Edge:** The end-point device containing sensors and actuators.

As IoT devices have increased in computing capability and extensively used to handle sensitive tasks and information, the industry has responded by providing secure methodologies and technologies to enhance trustworthiness and ensure correct function of the devices [7], [8]. These technologies involve both hardware and software, along with practices for design and development. This paper will survey some of these technologies, looking at existing solutions that provide *root-of -trust* and protect critical data such as private keys and embedded *Intellectual Property* (IP). Also, technologies that protect the operating device from malicious use will be surveyed, including architectural based methods of segregating critical and non-critical applications, regarding device storage, execution and memory. Software solutions will also be considered, particularly those that enhance segregation of potentially malicious applications from critical system components. However, while these technologies have relevance and offer protections to IoT devices, vulnerabilities continued to be found and exploited within IoT devices, with the security technology itself often being the weakness allowing access. Also, new vulnerabilities continue to be found, such as application and protocol implementation issues that inadvertently allow malicious use without any security technologies being aware.

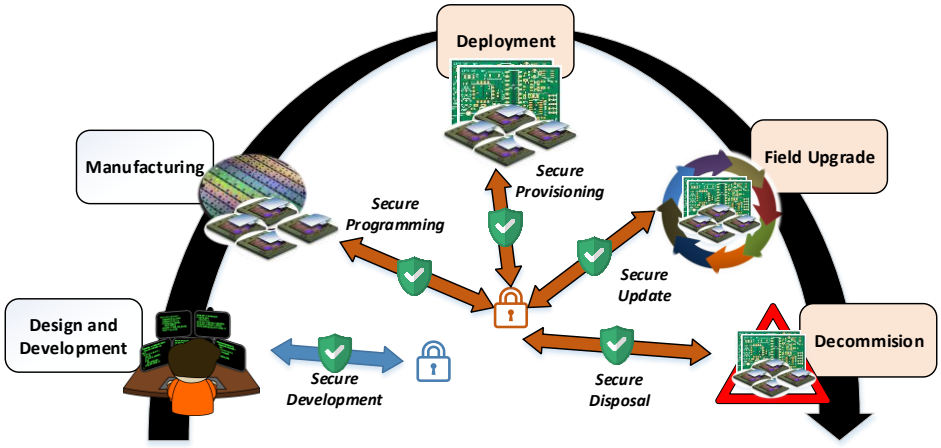

Figure 1: Secure embedded development life cycle of IoT devices.

To counter these issues and to complement existing security technologies, this paper introduces the concept of a *Security Policy Engine* (SPE), a hardware-based solution that monitors activities on the *System-on-chip* (SoC) communications bus. The proposed system utilises access control type policies to prevent unauthorised access to components by applications. The SPE is physically segregated from the operating system (OS) but can monitor activities such as sensor and actuator controls, application-specific access to peripherals, communication channels and other system resources using the system communication bus. The security policies are pre-defined policy comprising of two parts, (1) A whitelist of permitted access control and transaction rules used for system monitoring, and (2) A list of countermeasures, executed when a rule violation occurs. The SPE can co-operate with on-chip security components to initiate programmed attack responses to protect security critical assets. The proposed SPE is a hardware-based SoC security component designed for resource-constrained IoT devices.

2 Background

2.1 Secure IoT device life cycle

A concept of design for security has introduced to cater for the increasing complexity of IoT ecosystems and to attain a secure development life cycle [9], [10], [11]. A secure IoT device life cycle requires incorporation of security measures within every component, from securing the device, the data within the device and the communication channel between edge and the cloud service [12], [13], for the entire lifespan of the device, from *deployment* to *decommission*, as shown in Figure 1. The *design and development* stage should mitigate for security issues not yet known, allowing for capacity to add additional functionality after *deployment* or to conform with additional regulations after *deployment*. To achieve the desired device flexibility, the embedded designers have introduced hardware and software co-design approaches within the IoT ecosystem. The involvement of multiple designers and developers working at different hardware and software stacks increase the complexity of the ecosystem. One company may design a device, another supplies component software, another operates the network in which the device is embedded, and another deploys the device. A lack of clear security decisions and responsibilities opens IoT devices to a range of risks, vulnerabilities and attacks [14], [15]. The security challenge has further magnified by a lack of comprehensive, widely adopted standards for IoT security.

Life Cycle Stage	Hardware Security Requirements	Security Measures
Manufacturing	• Secure programming. • Circumvent device impersonation and overbuilding	• Unique device hardware identity • Bit-stream encryption • Certificate of conformance
Deployment	• Secure provisioning and management. • Secure configuration and control. • Secure monitoring and diagnosis	• Secure boot • Secure storage of secret data
Field Upgrade	• Secure firmware update to enforce and maintain chain-of-trust. • Firmware integrity to circumvent malicious software updates.	• Data integrity check mechanism • Authentication and authorisation mechanisms • Anti-tamper protection
Decommission	• Secure device disposal to ensure confidentiality of secret data. • Invalidate device and cannot be reinserted into the supply chain again.	• Secret data zeroization. • Revocation of certificate of conformance • Certificate revocation list (CRL)

Table 1: Classification of security requirements based on device life cycle.

The issue has been deemed serious enough that proposals to regulate IoT have been made at US Congress [16] and EU government levels [17]. Table 1 outlines the security requirements and measures to attain, deploy and ensure secure embedded development life cycle. These security requirements set the embedded design architectural choices to harness and enforce robust IoT security practices.

2.2 Root-of-trust (RoT) in IoT devices

To support, the security requirements listed in Table 1, *confidentiality* and *authentication* must adhere to the *root/trust anchor*. The *trust anchor* can be implemented using hardware and software approaches. Within the target device, the *root-of-trust* (RoT) implements the *trust anchor* [11], [18], [19]. The security device uses a *Proof-of-Knowledge* approach based on *challenge-response protocol*. The host authority initiates the verification process by:

- Generating a random challenge for the device.
- The device performs a cryptographic function using secret data and returns the output response.
- If the target device's output satisfies the challenge, the verifying authority treats the target as authentic.

When the concept of RoT first emerged, a software approach was used, with secret data stored in non-volatile memory as shown in Figure 2. The verifying authority read the secret data to confirm the authenticity of the device. However, this *software RoT* has exposed to device impersonation attacks.
The *trust anchor/root* must be baked within the device as a unique ID to circumvent this problem. Because, it is relatively difficult for an attacker to modify the functionality of the hardware to the extent where the hardware is immutable [6], [11], [20]. Therefore, *hardware root-of-trust* (HRoT) has introduced, as shown in Figure 2. It allows the device to generate a unique and unalterable identity by using a *Physically Unclonable Function* (PUF). PUF provides a unique challenge-response mechanism dependent on the complex and variable nature of the silicon material used to manufacture the device. By baking the *trust anchor* within the device, it provides isolated hardware-based verification of the target device and enables robust post-manufacturing device provisioning options to OEM and customers by generating PUF dependent obfuscated private keys (even unknown to chip manufacturers). The HRoT serves as device primary *root-of-trust* on which the remaining trust chain is built, maintained from system boot-up to power-down, improving firmware verification and integrity checks.

2.3 Trusted computing technologies and embedded security

The goal of *Trusted computing* is to develop technologies, which give users guarantees about the behaviour of the software running on their devices. A device can be trusted if it always behaves in an expected manner for the intended purpose even when the attacker gains control of the device [21]. It is a complex goal, covering security aspects that result

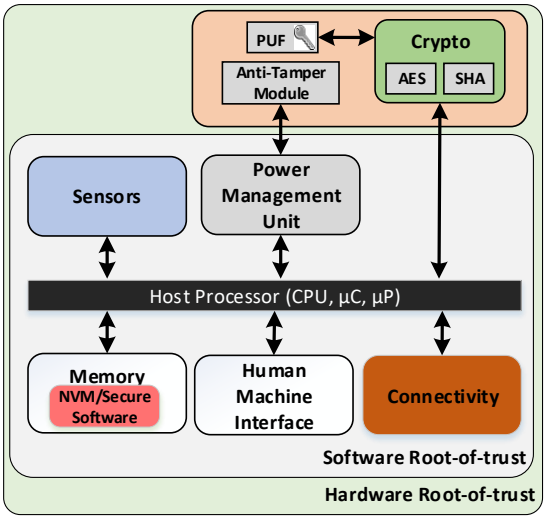

Figure 2: Software and Hardware Root-of-Trust

in a wide range of hardware and software co-design solutions. Many IoT designs make use of software portability, porting operating systems, network communication stacks, software libraries and service that were not designed primarily for secure or *machine-to-machine* (M2M) devices to reduce time-to-market and gain a competitive advantage. As a result, these devices might sub-optimally designed for their use case, which may impose a security risk. A similar trend is observed where third-party IP (usually encrypted netlist) is integrated to build secure hardware architectures and platforms which has opened doors to *hardware trojan* and *malware*. To address this issue while retaining benefits of software computing, hardware architects adopted the concept of *software virtualisation* and propagated it into hardware architectures. As a result, many embedded platforms can run operating systems (OS), such as Linux that significantly improving functionality and enabling for matured software-based security applications to further enhance device security. Support of *privileged* and *non-privileged* modes by OS provided a method to achieve isolation and segregation of system resources.
Intel *Software Guard Extensions* (SGX) is an example of an architecture extension that allows user level applications to make use of protected regions of memory, known as *enclaves*, to perform execution in isolation, with privacy from higher privileged applications, the OS, hypervisor, as well as drivers and device firmware. This therefore, allows running of trusted code within untrusted environments such as the cloud or device deployments. AEGIS, ARM TrustZone, TrustLite and Sanctum are other examples of architectures that support hardware isolation and allow execution of secure code within confined environment [22], [23], [24].
The *Trusted Execution Environment* (TEE), written by GlobalPlatform [25], is the secure area of the main processor that provides isolated execution of trusted application resources. The TEE is isolated from the *Rich Execution Environment* (REE) where the untrusted OS runs. The REE resources are accessible from TEE but not vice versa unless explicitly allowed by TEE. Some embedded solution providers have developed their own TEE solutions such as

Trustonics and Qualcomm QSEE. A TPM harnesses and maintains embedded *chain-of-trust* by authenticating devices and systems with the help of dedicated cryptographic hardware units. The TPM security functions are enforced and managed by the TEE, which runs on a virtualised hardware and shares the same system resources as the application. The ARM TrustZone is an example of such hardware-based system security architecture that virtually segregates the system resources into *secure* and *non-secure* worlds.
From the silicon and IP manufacturer's business point-of-view, embedded device security could be a feature that can be enabled and disabled on-demand or offered with optional levels of protection depending on the desired application use-case. To meet versatile IoT application requirements, it can be flexible on-demand hierarchical segregation of system resources into multiple *secure islands* that can only exchange data within their designated *secure island*.

2.4 Trusted computing-based solutions

Samsung Knox is a security suite developed for mobile devices that employs *trusted computing*. Knox features a layered array of features, presented to the user as a secure environment for segregating work and personal activities. Bootloader protection verifies the integrity of the secure world OS and the kernel. Security Enhancements for Android to prevent unauthorised application access, based on confidentiality requirements. *TrustZone-based Integrity Measurement Architecture* (TIMA) monitors the kernel to detect compromise and corruption from within the TrustZone. The attestation feature to remotely validate the device's authenticity before access. Knox makes use of ARM TrustZone, providing a TEE platform for securing the contained work environment and associated applications, increasing security within *Bring your own device* (BYOD) deployments [26].

2.5 Software-based IoT security solutions

The Android *Application Sandbox* is a method used by the Android OS to segregate applications from one another, thus preventing one application from being able to access data from another application, except under controlled circumstances. Android makes use of *Security-Enhanced Linux* (SELinux) to enhance the *Application sandbox* and ensure application permissions, as configured by the user, are adhered to [27].
SELinux is an access control layer, implemented within the Kernel, enforcing policies to ensure minimal privileges are granted to applications, to limit their function only to how they are specified. Within a SELinux environment, an exploited vulnerability will be unable to gain access beyond the SELinux boundaries, as defined by the administrator, enforcing the principle of least privilege [28]. Designed by the NSA and RedHat, SELinux has widely used within commercial Linux deployments.
Software containerisation implements a form of application isolation, under the same kernel instance. *Containers* allow for segregation of applications while allowing access to OS components that have been assigned access, with kernel measures to limit resource consumption. Due to the usage of the system's native kernel, *containerisation* does not apply overheads that virtualisation consists of. *Containers* have been adopted for use in mobile applications by several vendors including Samsung (Knox), IBM (MaaS360), Blackberry (Secure Work Space) and Google (Android for Work) [29], [26], [30].

3 Vulnerabilities and threats posed to IoT

3.1 IoT relevant security issues and root causes

IoT devices may be attacked and compromised by a wide range of methods, including the same as those used against cloud and regular IT systems along with new attack vectors created by the IoT. These attacks may target a wide range of aspects, from the protocol stack to the systems' applications, bootloader and hardware peripherals, including even the security technologies designed to protect them. The following examples specify some common design vulnerabilities affecting IoT, along with their root causes that enable them to be use against IoT devices. This section will survey embedded and IoT device hardware and software security vulnerabilities and attacks reported in open literature.

Software configuration: Perhaps the best attack to affect IoT devices is the Mirai Botnet, which affected network connected webcams and DVRs, amongst other devices. Taking advantage of poor device configuration practices by the IoT OEM, this attack scanned the internet, compromising the IoT devices using a standard password and open telnet port, peaked at 600,000 infections. Upon installing a basic malware application, it made use of IoT's network connectivity to launch *Distributed Denial-of-Service* (DDOS) attacks against online hosting services, including OVH and DynDNS, peaking at a record 1.2Tb/s [4].

Software development: Use of vulnerable libraries and components: The OpenSSL Heartbleed CEV-2014-0160 is a prime example. This vulnerability consisted of a missing boundary check within a *memcpy* function, allowing an attacker to view arbitrary data from the OS memory. Vulnerabilities of this nature are of serious risk to IoT as their smaller memory size eases the difficulty in locating sensitive data [31].

Key management practices: Another attack leveraged key management vulnerabilities in the ZigBee IoT wireless protocol and *side-channel analysis* of device to extract a common firmware-signing key, to spread malware using ZigBee as the communication medium. The firmware image-signing key could be extracted from the device using correlation power analysis, allowing modified malicious firmware to be installed. This compromised device could propagate to nearby devices, taking advantage of a leaked universal key for the ZigBee protocol, compromising them with the same malicious firmware update [32].

User interface: When interface methods are used to protect the device and user, they may become an attack surface. Vulnerabilities have been found to exist in mobile and touchscreen devices, (CVE-2015-3860) allowing bypass of the lock screen, as well as theft based "Factory Reset Protection" on Android devices. These methods involve taking advantage of sophisticated navigation structures and unprotected text entries, allowing a malicious user to escape locked environment [33].

Kernel and driver: These issues are a source of major vulnerabilities in embedded devices, due to the low-level access these components require. Vulnerabilities such as (CVE-2016–5195) Dirty COW (Copy on write), exploit a race condition found in kernel memory subsystem and handling of read-only memory mappings. An unprivileged user could exploit this with a simple program to gain write access to read-only memory [34].

Hardware-based: ARM TrustZone was shown to lack proper rollback protection in the *secure boot*. Using an old key, an attacker could load a malicious application into the TEE to run in the secure world, providing access to the device's secret data [20]. Mitigation of this attack type is to formulate a well-defined rollback and update policy. Qualcomm's QSEE and Samsung Exynos devices have been found vulnerable to similar downgrade and rollback attacks [35]. Chen *et al*. further compromised ARM TrustZone cache. Using side channel analysis, the attackers were successfully able to monitor cache activity of Android's cryptographic operations [36]. Mitigation of cache-based attacks consists of deploying different caches or to clear the cache when switching from secure to non-secure world. Frédéric has launched attack against an Amlogic S905 processor that uses ARM TrustZone for DRM & other security features. He successfully managed to bypass secure boot and break the chain-of-trust deploying series of attacks [37].

Insider and user error: Finally, it should not be understated that the user can inadvertently or deliberately cause system vulnerabilities [38].

3.2 Shortcomings of existing solutions

After reviewing different aspects of security problems in IoT devices, their security methodologies and architectures to protect secret assets. It is clear that these existing approaches fall short of their desired security goals. This due to:

- A complex IoT eco-system where different vendors are developing hardware, software and security components, leading to interoperability issues and vulnerabilities in hardware, software and protocol stack boundaries.
- Development of inconsistent and vulnerable software applications, libraries and protocols, caused by lack of security aware design and development practises.
- A lack of IoT security standards which vendors can incorporate to improve their product development life cycle to reduce root cause of vulnerabilities.

Though reliance on robust and secure software development practices focusing on application, kernel and device-driver layers alone is not sufficient. Since vulnerabilities may be found and exploited in other areas outside of the developer's control, such as a CPU vulnerability, a flaw within a common standard, or other unforeseen failures. Additionally, existing *trusted computing* technologies may later be compromised, if the components it relies upon are found to be vulnerable, as has been shown with Intel SGX and ARM TrustZone technology. Finally, it is important to mention that even when the devices are operating correctly, they may be compromised due to human operational errors, be that a user misconfiguration, error, exposure of authentication through a social engineering attack, or insider threat.

4 Embedded policing and policy enforcement approach

To approach the identified shortcomings in this paper, we propose a *policing* and *policy enforcement* approach, to enhance system security and scalability by complementing existing security technologies and embedded architectures. Based on hardware, but complementing software concepts such as SELinux, this policing approach follows the principle of least privilege, monitoring system communication and behaviour of the system resources and comparing it to authorised behaviours for that resource. In case of unexpected behaviour, for example, a malicious action from a compromised application or launching of attack by the adversary, the system can take active and passive counter-measures to mitigate the attack and protect the system assets. Like SELinux, the proposed policy approach aims to limit an attack's capability, assuming other system protections have failed. Expected behaviours shall be defined as the system *security policy*, which can be updated at any stage of the device life cycle. The policing approach, deployed at the system communication layer, will be transparent to existing embedded system architectures that use the de-facto industry standard ARM AMBA-AXI4 SoC bus communication protocol, along with their software stacks (bare-metal, embedded Linux, Real-time operating system (RTOS), Hypervisor, device drivers etc.). The proposed policing and policy enforcement system architecture will not only enhance the security of the device but will also improve its flexibility to meet changing security requirements of next-generation embedded and IoT architectures, as well as the next-generation of threats.

4.1 Embedded policy platform architecture

Embedded systems are complex, with system designers and software developers often required to consolidate different functionalities into a single application by combining sensitive and non-sensitive data. Heterogeneous *multiprocessor system-on-chip* (MPSoC) platforms composed of multiple processors, hardware IPs, on-chip support of major peripheral interfaces and shared memory resources, are a suitable choice to achieve design goals, as this aids adaptability, reusability, and upgradability and reduces time-

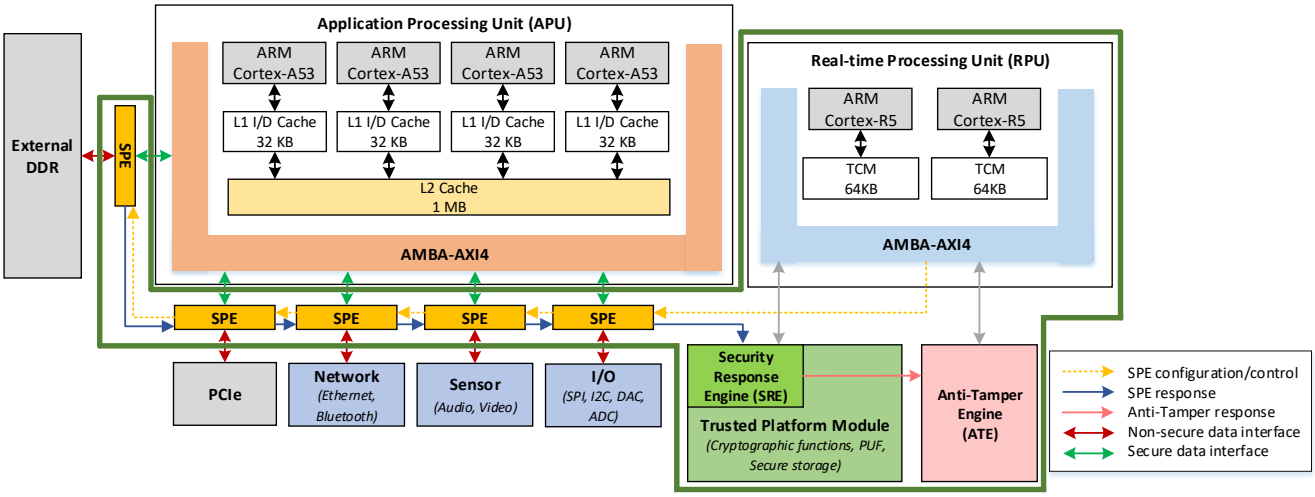

Figure 3: Block diagram of system architecture to implement the proposed policing and policy enforcement approach. It allows to harness and enforce defined security policies within FPGA MPSoC platform.

to-market. They offer on-chip hardware security features to protect the critical assets and attain security that is essential requirements of next-generation secure systems. To realise the proposed policing and policy enforcement approach, we present a system architecture that uses Xilinx Zynq UltraScale+ MPSoC as shown in Figure 3. This platform has chosen due to the on-chip availability of hardware security and cryptographic features, essential for building HRoT.

The MPSoC features an on-chip Quad-core ARM Cortex-A53 *Application Processing Unit* (APU) for general purpose computing and a Dual-core ARM Cortex-R5 *Real-time Processing Unit* (RPU) for critical applications requiring deterministic low-latency. The APU supports multiple software stack configurations based on application design requirements. It can be configured to execute applications as bare metal, Linux, hypervisor software stacks [39]. It has an ARM TrustZone hardware module that extends the secure and non-secure world applications to ARM AMBA-AXI4 bus transaction using non-secure (NS) bit of AxPROT and limits access to the system resources. In proposed system architecture, it has assumed that user software will execute on APUs and treat it as *untrusted application* host processor that utilises system data and memory resources. On the other hand, the application to configure and manage the critical components of our proposed system architecture will run on RPU and be treated as *trusted application*. RPU has a dual 64KB *tightly-coupled memory* (TCM) that is physically isolated from APU, L1 and L2 cache memory. This system security boundary has highlighted in Figure 3. This physical isolation and segregation ensure that *untrusted applications* cannot read, write or modify any part of TCM, overcoming one of the shortfalls of virtualised security architectures. Central to our policing and policy enforcement approach are following hardware engines:

1. Security policy engine (SPE)
2. Security response engine (SRE)
3. Anti-tamper engine (ATE)

The proposed SPE has implemented using FPGA glue logic to monitor system-level communication among system peripherals and memory blocks, comparing against the defined policies for each resource, and initiates pro-active response actions to counteract attacks. Therefore, it can be ported to other FPGA and ASIC architectures as IP and is not limited to the proposed platform. To maximise flexibility to changes in security policies, each hardware engine has AXI4-Lite communication, that can be used by the RPU to configure and program engine specific security parameters. This can fulfil the proposed architecture to meet the dynamic security requirements of different devices.

The ARM AMBA-AXI4 protocol specification specifies that the bus establishes the communication between a master and a slave using five separate channels; *address write*, *write*

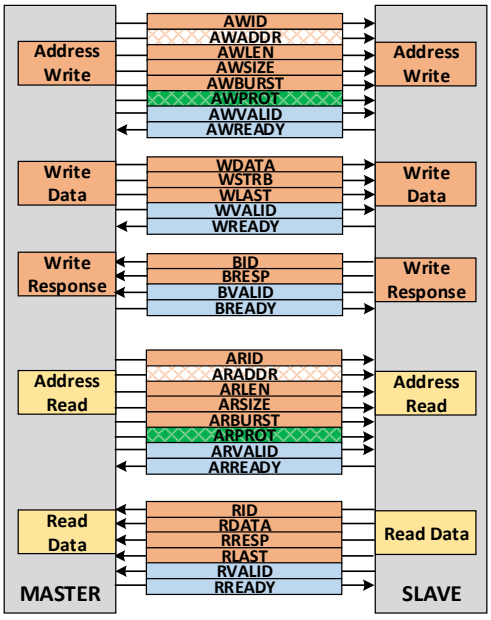

Figure 4: ARM AMBA-AXI4 compliant master-slave communication interface signals

data, *write response*, *address read* and *read data* as illustrated in Figure 4. Each channel has VALID and READY handshake signals. VALID indicates that data available on the bus is valid, and READY indicates that the receiver is ready to receive the data. To exchange data, both VALID and READY signals should be active. Both address channels have 3-bit AxPROT signals, where the *non-secure* (NS) bit indicates the security attribute of the transaction.

4.2 Security policy Engine (SPE)

The idea of the SPE is comparable to a policy-based hardware firewall, which checks the approved list of devices, and either grants or restrict access to the device. The SPE engine enforces system resource specific requirements by actively monitoring shared communications at the interface between AMBA-AXI4 bus and target system resource (memory, sensor, actuator etc.) The SPE is capable of independently monitoring and enforcing policies related to that resource. The SPE itself is composed of an *AXI4 sniffer block*, *Device table*, *Policy table*, *Decision block and Engine configuration block* as shown in Figure 5.

4.2.1 SPE Hardware Components

AXI4-Sniffer: This component monitor and maintain the necessary AXI4 handshake signals as defined in ARM AMBA-AXI4 communication protocol specification, until the *decision block* checks and grants access to the target system resource as shown in Figure 5. This component is essential to maintain correct system level communication.
The AXI4-sniffer receives AXI-transaction from the shared system communication bus and samples *requesting resource* and *target resource* addresses. It samples AWVALID and ARVALID control signals to identify whether the *requesting resource* has asked for reading or writing operation on the *target resource*. It samples the target address, AWADDR or ARADDR, and forwards it to the *policy table*. The resource address is passed to the *device table* to locate the base-address of the policies stored in the *policy table* as shown in Figure 5. During this process, to maintain physical isolation between *requesting resource* and the *target resource*, the AXI4-sniffer physically blocks the handshake signals by de-asserting WVALID and RVALID control signals. Once the *decision block* validates and allows access, based on the defined policy, the AXI4-sniffer asserts the WVALID or RVALID signals respectively.

Device table: This contains the list of trusted *requesting resources* that are permitted to access the *target resource*. In this table, each trusted *requesting resource* has a corresponding base-address of the security policies that specify the access rights linked to it, as shown in Figure 5. The length of the *device table* is limited to the number of *requesting resources*. It can be realised by distributed RAM that uses LUT resources of FPGA logic to extract *policy table* address. The *device table* allows not only to filter trusted and un-trusted *requesting resources* but can be used to detect intrusions. If an intrusion is detected, it can be suppressed by initiating a pro-active mitigation response by SRE to secure the system's assets.

Policy table: This contains the permission rights of the *target resource* that corresponds to each trusted *requesting resource*. The stored policies can be simple as address read/write permissions or complex policies to check variable length (AxSIZE and AxLEN) transfers, security profile assigned by the software (AxPROT), or quality of service (AxQOS). It can be realised by block RAM (BRAM) in FPGA logic. From a software point-of-view, the *policy table* not only provides means to secure assets, it also provides a mechanism that can be used for secure provisioning of the device at any stage of the device life cycle, by defining role-based access policies.

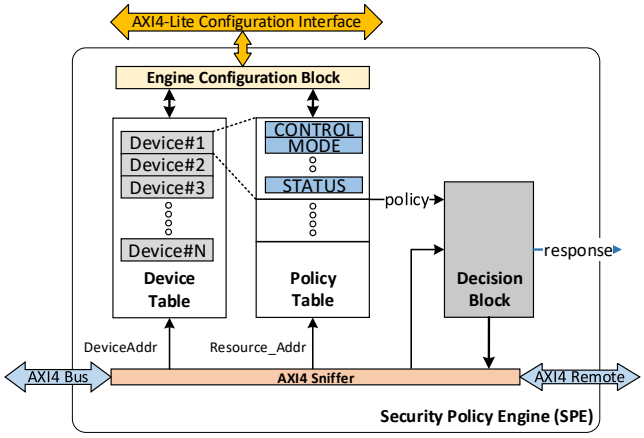

Figure 5: Block diagram of Security Policy Engine (SPE)

Decision block: This receives the programmed/reference policy issued by the *policy table*, compares the status of AXI4 signals, as per security policy and generates a result. The generated result will be passed to *AXI-Sniffer*, which then either grant or block the requested AXI4 transaction as shown in Figure 5.

Engine configuration block: This provides functionality to program and configure the security parameters of the SPE, using a secure application running on the RPU via the AXI4-Lite communication interfaces, as shown in Figure 5. It allows programming and updating of both the *device table* and the *policy table*.

4.3 Security Response Engine (SRE)

This engine provides effective active responses to security threats reported by the *decision block*, to initiate mitigation response. It does this by interrupting the *secure code* running on the RPU and requests the time-critical interrupt service. The *pro-active* response shall be initiated by executing the *interrupt service routine* (ISR) of the SRE and is enforced by the ATE. Following are examples of pro-active responses that can be enforced:

MPSoC	FPGA fabric (technology)	Speed Grade	Area				Frequency (MHz)
			LUT	LUTRAM	FF	BRAM	
Zynq-7000	Kintex-7 (28nm)	-3	87	32	304	1	250
Zynq UltraScale+ EG	Kintex UltraScale+ (16nm)	-3	55	32	235	1	333

Table 2: Synthesis results of proposed security policy engine (SPE) on Zynq-7000 and Zynq UltraScale+ MPSoC

- Erasure of secret keys stored in the device to ensure confidentiality.
- Permanent disabling of the system cryptographic functions, if the system has compromised.
- Disabling system read-back interfaces.
- Initiation of secure lockdown of the system resources.
- Initiation of system reset.

4.4 Anti-tamper engine (ATE)

This engine enforces the pro-active measures initiated by the SRE. Also, it provides passive physical security measures against hardware level attacks, such as side- channel analysis, by monitoring the devices physical parameters such as voltage and temperature. An alarm can be initiated if the system violates the set conditions. Following are the anti-tamper features that can be ensured:

- Maintain uninterruptable internal clock sources.
- Monitoring on single-event-upset to ensure fault tolerant execution during operational device life cycle.
- Voltage and temperature monitoring to ensure device operation and resist against active hardware attacks.

4.5 Area and timing results

The proposed *security policy engine* (SPE) and its hardware components presented in Section 4.2 are coded in Verilog HDL. The design was simulated and synthesised using Xilinx Vivado v2017.1 on the high-end chips of the Zynq-7000 and Zynq UltraScale+ MPSoC family. The area and timing results are reported in Table 2. The *device table* implemented using Distributed RAM utilises 32 LUTRAM, while the *policy table* used a BRAM. The design is compact and consumes less than 1% of FPGA glue logic available on Zynq chips. The reported timing results shows that SPE can operate at maximum 250 MHz and 333 MHz on Kintex-7 and Kintex UltraScale+ chips. The reported timing is limited by the maximum possible speed of ARM AMBA-AXI4 bus on these chips [40], [41].

5 Preliminary Use-case

A heterogeneous system composed of BRAM based memory devices has realised as shown in Figure 6. It has considered that specific location of each device holds secret assets that shall be protected to ensure confidentiality. Each device is connected to the host through the ARM AMBA-AXI4 communication bus via an AXI4-interconnect. To integrate the proposed approach, the SPE has integrated between the device and the shared AXI4 bus. It polices the addressed transactions, checks the read and write access policy attached to the *requesting resource* (BRAM memory location) and either grants or refuses access to the asset as shown in Figure 6. In this use case, a bare-metal software application approach has been adopted to avoid software development complexity. At power-up, the host application configures the platform by programming the device and policy table of each SPE through the AXI4-Lite interface as shown in Figure 6. The *policy table* stores the register read and write access permissions for each device.

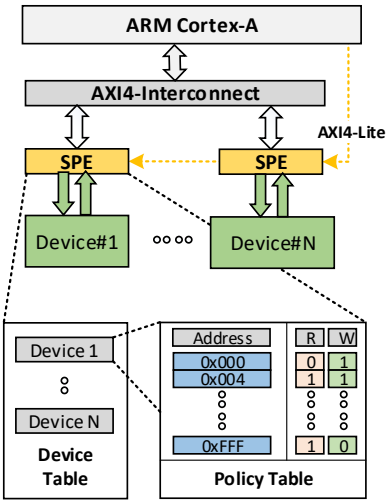

Figure 6: Read/write policy enforcement on simple memory devices within MPSoC.

The presented system is synthesized and implemented on the widely available Avnet Zedboard (Zynq-7000) and Xilinx ZCU102 (Zynq UltraScale+) evaluation boards using Xilinx Vivado v2017.1. Table 3 reports the preliminary implementation results. The BRAM utilisation of both implementations is consistent. Each SPE has used single BRAM each to implement policy table, while each memory device consumed two BRAMs. Due to technology scaling and

Development Board	FPGA fabric (technology)	Speed Grade	Area Utilisation				Frequency (MHz)
			LUT	LUTRAM	FF	BRAM	
Zedboard	Artix-7 (28nm)	-1	1043	91	1436	6	164
ZCU102	Kintex UltraScale+ (16nm)	-2	2750	116	3354	6	302

Table 3: Implementation results of proposed security policy engine (SPE) on Zedboard and ZCU102

better silicon (speed grade), ZCU102 implementation achieved ≈ 1.84 times better timing than Zedboard. However, it comes at the cost of ≈ 2.63 and 2.33 times more LUT and FF utilisation required by the system infrastructure (AXI-Interconnect, Reset processing system etc.) essential for Zynq UltraScale+ MPSoC.

6 Conclusion and future work

The paper has presented preliminary proof-of-concept work. The proposed policing approach will not only help to tackle security problems for future embedded devices, but also existing hardware architectures that did not consider security as a design requirement in their design flow. The integration of a software-defined and hardware-enforced *security policy engine* (SPE) brings adaptability to security platforms, where the same platform can be used to fulfil the diverse security requirements of different application domains and security levels. Additionally, the proposed system architecture compliments state-of-art chain-of-trust and authentication mechanisms, which mitigates against existing vulnerabilities and attacks and provides robust security measures to circumvent new vulnerabilities and attack vectors.
The presented work will be continued to enable secure configuration of system hardware components and policing of more complex communication attributes. From a software perspective, future work shall be targeted to provide isolation and segregation of hardware resources requested by software components, while being managed by the operating system.

References

[1] D. Evans, "The Internet of Things: How the Next Evolution of the Internet is Changing Everything," *Cisco Internet Business Solutions Group (IBSG)*, Cisco, https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf, (2011).

[2] R. v. d. Meulen, "Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016," *Gartner*, https://www.gartner.com/newsroom/id/3598917, (2017).

[3] Y. Yang *et al*, "A Survey on Security and Privacy Issues in Internet-of-Things," *IEEE Internet of Things Journal*, **vol. 4,** pp. 1250-1258, (2017).

[4] Antonakakis, Manos and April, Tim and Bailey, Michael and Bernhard, Matt and Bursztein, Elie and Cochran, Jaime and Durumeric, Zakir and Halderman, J Alex and Invernizzi, Luca and Kallitsis, Michalis and others, "Understanding the Mirai Botnet," *USENIX Security Symposium*, .

[5] A. Rodriguez-Mota *et al*, "Towards IoT cybersecurity modeling: From malware analysis data to IoT system representation," in *2016 8th IEEE Latin-American Conference on Communications (LATINCOM)*, pp. 1-6, (2016).

[6] A. Sengupta and S. Kundu, "Guest Editorial Securing IoT Hardware: Threat Models and Reliable, Low-Power Design Solutions," *IEEE Transactions on very Large Scale Integration (VLSI) Systems*, **vol. 25,** pp. 3265-3267, (2017).

[7] C. Lesjak, D. Hein and J. Winter, "Hardware-security technologies for industrial IoT: TrustZone and security controller," in *41st Annual Conference of the IEEE Industrial Electronics Society (IECON)*, (2015).

[8] H. He *et al*, "The security challenges in the IoT enabled cyber-physical systems and opportunities for evolutionary computing & other computational intelligence," in *IEEE Congress on Evolutionary Computation (CEC)*, pp. 1015-1021, (2016).

[9] H. Khattri, N. K. V. Mangipudi and S. Mandujano, "HSDL: A security development lifecycle for hardware technologies," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 116-121, (2012).

[10] S. Sicari *et al*, "Security, privacy and trust in Internet of Things: The road ahead," *Computer Networks*, **vol. 76,** pp. 146-164, (2015).

[11] G. Chopra, R. Kumar Jha and S. Jain, "A survey on ultra-dense network and emerging technologies: Security challenges and possible solutions," *Journal of Network and Computer Applications*, **vol. 95,** pp. 54-78, (2017).

[12] B. Zhang *et al*, "The cloud is not enough: Saving IoT from the cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'15)*, (July, 2015).

[13] F. Bonomi *et al*, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, Helsinki, Finland, pp. 13-16, (2012).

[14] A. R. Sadeghi, C. Wachsmann and M. Waidner, "Security and privacy challenges in industrial internet of things," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1-6, (2015).

[15] M. M. Hossain, M. Fotouhi and R. Hasan, "Towards an analysis of security issues, challenges, and open problems in the internet of things," in *2015 IEEE World Congress on Services*, pp. 21-28, (2015).

[16] S. Warner and M. R., "To provide minimal cybersecurity operational standards for Internet-connected devices purchased by Federal agencies, and for other purposes." Senate Bill: Homeland Security and Governmental Affairs, https://www.congress.gov/bill/115th-congress/senate-bill/1691/text, (2017).

[17] ENISA, "Infineon – NXP – STMicroelectronics – ENISA Common Position On Cybersecurity," https://www.enisa.europa.eu/publications/enisa-position-papers-and-opinions/infineon-nxp-st-enisa-position-on-cybersecurity, (2015).

[18] T. Abera *et al*, "Invited - things, trouble, trust: On building trust in IoT systems," in *Proceedings of the 53rd Annual Design Automation Conference*, Austin, Texas, pp. 121:6, (2016).

[19] G. Tuna *et al*, "A survey on information security threats and solutions for Machine to Machine (M2M) communications," *Journal of Parallel and Distributed Computing*, **vol. 109,** pp. 142-154, (2017).

[20] U.S Department of Homeland Security, "Strategic principles for securing the Internet of Things (IoT)," *Guidance Publication:*, https://www.dhs.gov/securingtheIoT, (2016).

[21] P. Maene *et al*, "Hardware-Based Trusted Computing Architectures for Isolation and Attestation," *IEEE Transactions on Computers*, vol. PP, pp. 1, (2017).

[22] G. E. Suh *et al*, "AEGIS: Architecture for tamper-evident and tamper-resistant processing," in *Proceedings of the 17th Annual International Conference on Supercomputing*, San Francisco, CA, USA, pp. 160-171, (2003).

[23] P. Koeberl *et al*, "TrustLite: A security architecture for tiny embedded devices," in *Proceedings of the Ninth European Conference on Computer Systems*, Amsterdam, The Netherlands, pp. 10:14, (2014).

[24] V. Costan, I. Lebedev and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 857-874, (2016).

[25] GlobalPlatform, "The Trusted Execution Environment: Delivering Enhanced Security at a Lower Cost to the Mobile Market," *White Paper*, https://www.globalplatform.org/documents/GlobalPlatform_TEE_White_Paper_Feb2011.pdf, (2011).

[26] Samsung Electronics, "In-Depth Look at Capabilities: Samsung KNOX and Android for Work," *White Paper*, https://kp-cdn.samsungknox.com/759d1af2cb26204a2f4852b151d9dc9c.pdf, (2015).

[27] Google, “System and kernel security”, https://source.android.com/security/overview/kernel-security#the-application-sandbox Security overview document, (2017).

[28] M. B. Gregory and A. S. Reninger, "Teaching SELinux in introductory information assurance classes," in *42nd Hawaii International Conference on System Sciences (HICSS)* , Big Island, HI, pp. 1-8, (20 Jan, 2009).

[29] T. Meng, Z. Shang and K. Wolter, "An empirical performance and security evaluation of android container solutions," in *International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, London, UK, pp. 1-8, (19 Oct, 2017).

[30] BlackBerry, "Secure Work Space for iOS and Android," *White Paper*, https://uk.blackberry.com/content/dam/blackBerry/pdf/business/english/bes10/BES10-2-SWS-EMM-data-sheet.pdf, (2013).

[31] I. Ghafoor *et al*, "Analysis of OpenSSL heartbleed vulnerability for embedded systems," in *17th IEEE International Multi Topic Conference (INMIC)*, pp. 314-319, (30 Apr, 2015).

[32] E. Ronen *et al*, "IoT goes nuclear: Creating a ZigBee chain reaction," in *IEEE Symposium on Security and Privacy (SP)*, pp. 195-212, (26 June, 2017).

[33] National Institute of Standards and Technology, (NIST), "CVE-2015-3860 Detail," *Software Vulnerability*, https://nvd.nist.gov/vuln/detail/CVE-2015-3860, (2015).

[34] A. P. Saleel, M. Nazeer and B. D. Beheshti, "Linux kernel OS local root exploit," in *IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pp. 1-5, (8 Aug, 2017).

[35] Y. Chen *et al*, "Downgrade Attack on TrustZone," *Computing Research Repository (CoRR)*, vol. abs/1707.05082, (2017).

[36] M. Lipp *et al*, "ARMageddon: Cache attacks on mobile devices," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 549-564, (2016).

[37] Fred, "Amlogic S905 SoC: bypassing the (not so) Secure Boot to dump the BootROM,", http://www.fredericb.info/2016/10/amlogic-s905-soc-bypassing-not-so.html, (2016).

[38] F. L. Greitzer *et al*, "Analysis of unintentional insider threats deriving from social engineering exploits," in *IEEE Security and Privacy Workshops (SPW)*, San Jose, CA, pp. 236-250, (20 Nov, 2014).

[39] Xilinx, "Zynq UltraScale+ MPSoC Embedded Design Methodology Guide," *UG1228 (V1.0)*, Xilinx Inc.,https://www.xilinx.com/support/documentation/sw_manuals/ug1228-ultrafast-embedded-design-methodology-guide.pdf, (2017).

[40] Xilinx, "Zynq-7000 All Programmable SoC (Z-7030, Z-7035, Z-7045, and Z-7100) : DC and AC Switching Characteristics," *DS191 (V1.18)*, https://www.xilinx.com/support/documentation/data_sheets/ds191-XC7Z030-XC7Z045-data-sheet.pdf, (2017).

[41] Xilinx, "Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics," *DS925 (V1.9)*, https://www.xilinx.com/support/documentation/data_sheets/ds925-zynq-ultrascale-plus.pdf, (2017).